Scalable Parallel Trace-Based Performance Analysis

Introduction

To satisfy their increasing demand for computing power, advanced numerical simulations are required to harness larger numbers of processors offered by modern capability computing systems, such as the IBM BlueGene/L system JUBL at Forschungszentrum Jülich. Unfortunately, satisfactory speedup on many thousands of processors is extraordinarily hard to achieve. Sustained application performance is often significantly below the theoretical limit and leaves substantial room for optimization. However, tools that normally assist developers in the optimization process cease to work in a satisfactory manner when deployed on large processor counts.

Event tracing has been a well-established technique for post-mortem performance analysis of parallel applications. Time-stamped events, such as entering a function or sending a message, are recorded at runtime and analyzed afterwards with the help of software tools. In this context, automatic off-line trace analyzers, such as the EXPERT tool from the KOJAK toolset [1], can conveniently provide relevant information by automatically searching traces for complex patterns of inefficient behavior and quantifying their significance. In addition to usually being faster than a manual analysis, this approach is also guaranteed to cover the entire event trace and not to miss any pattern instances.

However, as the size of parallel systems and the number of processors used by individual applications rises, the traditional approach of sequentially analyzing a single global trace file becomes increasingly constrained by the large number of events. A new project aimed at overcoming these limitations, SCALASCA [2], started at the beginning of 2006 in a Helmholtz-University Young Investigators Group established between Forschungszentrum Jülich and RWTH Aachen University.

In this article, we outline how the pattern search can be done in a more scalable way by exploiting both distributed memory and parallel processing capabilities available on modern large-scale systems and discuss first empirical results. For a more detailed discussion, the interested reader may refer to [3].

Parallel Analysis Approach

Instead of sequentially analyzing a single and potentially large global trace file, we analyze multiple local trace files in parallel based on the same parallel programming paradigm as the one used by the application under investigation. For simplicity, we currently have restricted ourselves to handle only single-threaded MPI-1 applications. The analyzer, which is an MPI application in its own right, is executed on as many CPUs as the target application. This allows the user to run it after the target application within a single batch job, which avoids additional waiting time in the batch queue. The parallel analyzer uses a distributed memory approach, where each process reads only the local trace data that were recorded for the corresponding process of the target application. This addresses scalability specifically with respect to larger numbers of processes. Since the size of local traces can be limited by selective tracing – i.e., by recording events only for code regions and time intervals of particular interest – we assume that the local trace data can be completely held in the main memory of the compute nodes. This has the advantage of having efficient random-access to individual events, whereas this is often not the case when dealing with a global trace file.

The actual analysis can then be accomplished by performing a parallel replay of the application's communication behavior. The central idea behind this approach is to analyze a communication operation using an operation of the same type. For example, to analyze a point-to-point message, the event data necessary to analyze this communication is also exchanged in point-to-point mode between the corresponding analysis processes. To do this, the new analysis traverses local traces in parallel and meets at the synchronization points of the target application by re-enacting the original communication.

The replay-based analysis approach can be used to search for a large number of inefficiency patterns. Our current prototype supports all but one rarely significant MPI-1 pattern offered by the original sequential EXPERT tool. Two examples of these patterns are diagrammed in Figure 1. Their detection algorithms will be used to illustrate the parallel analysis mechanism below.

As an example of inefficient point-to-point communication, consider the so-called Late Sender pattern. Here, a receive operation is entered by one process before the corresponding send operation has been started by the other. The time lost is therefore the difference between the timestamps of the enter events of the MPI function instances which contain the corresponding message send and receive events. The complete Late Sender pattern consists of four events, specifically the two enter events and the respective message send and receive events.

During the parallel replay, the detection of this performance problem is triggered by the point-to-point communication events involved (i.e., send and receive). That is, when a send event is found by one of the processes, a message containing this event and the associated enter event is created. This message is then sent to

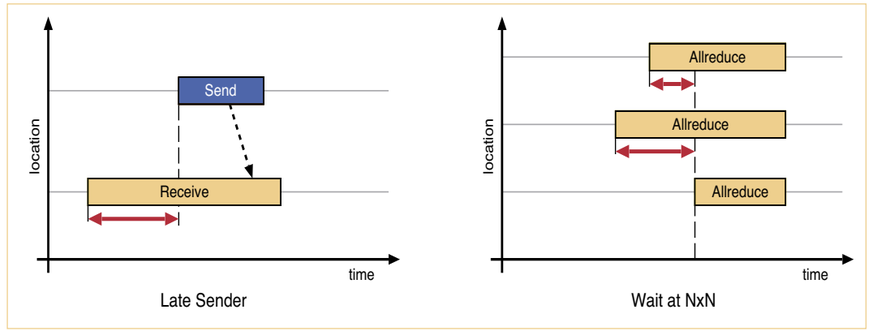

Figure 1: Two examples of inefficient program behavior; one for point-to-point communication (Late Sender) and one for collective operations (Wait at N x N)

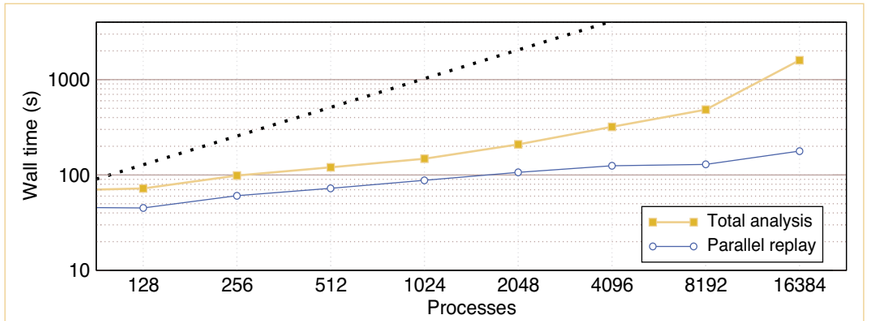

Figure 2: Wall-clock execution times for SMG2000 analysis using the new prototype at a range of scales. Linear scaling is the bold dotted line

the process representing the receiver using a point-to-point operation. To ensure the correct matching of send and receive events, equivalent tag and communicator information are used to perform the communication.

When the receiver reaches the receive event, the aforementioned message containing the remote constituents of the pattern is received. Together with the locally available constituents (i.e., the receive and the enter events), a Late Sender situation can be detected by comparing the timestamps of the two enter events and calculating the time spent waiting for the sender.

The second important type of communication operations are MPI collective operations. As an example of a related performance problem, consider the detection of the Wait at N x N pattern, which quantifies the waiting time due to the inherent synchronization in N-to-N operations, such as MPI_Allreduce.

While traversing the local trace data, all processes involved in a collective operation will eventually reach their corresponding collective exit events. After verifying that it relates to an N-to-N operation, accomplished by examining the associated region identifier, the analyzer invokes the detection algorithm, which determines the latest of the corresponding enter events using an MPI_Allreduce operation. After that, each process calculates the local waiting time by subtracting the timestamp of the local enter event from the timestamp of the enter event obtained through the reduction operation. The group of ranks involved in the analysis of the collective operation is easily determined from the communicator of the original collective operation.

Results

To evaluate the effectiveness of parallel analysis based on a replay of the target application's communication behavior, a number of experiments with our current prototype implementation have been performed at a range of scales. Measurements were taken on the 8-rack IBM BlueGene/L system JUBL using a dedicated partition consisting of all of the compute nodes for the parallel analyses.

Figure 2 charts wall-clock times for the analysis of ASC benchmark SMG2000 traces with a range of process numbers (the 8-fold doubling of process numbers necessitates a log-log scale

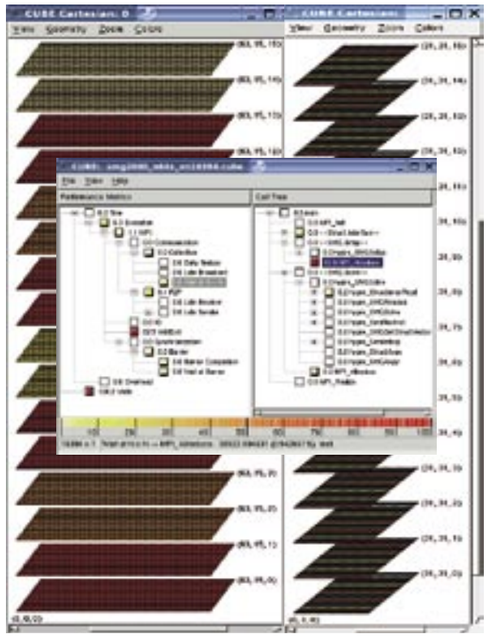

Figure 3: Analysis report for ASC SMG2000 on 16,384 processors of BlueGene/L highlighting the distribution of the Wait at N x N performance metric on the physical machine topology distribution (left) and MPI process topological distribution (right) for a particular call path

to show the corresponding range of times). The figure shows the total time needed for the parallel analysis (including trace reading and writing a complete analysis report) and the time taken by the parallel replay itself without file I/O. Due to the often considerable variation in the time for file I/O (e.g., depending on overall file-system load) the times reported are the best of several measurements.

The parallel replay for the largest set of execution traces from 16,384 SMG2000 processes, amounting to over 40,000 million events (230 GBytes of trace files), took less then 3 minutes. With the latest improvements for merging the analysis results (in comparison to [3]), which is reflected in the curve showing the total analysis time, the full analysis for 16,384 processes completed in less than 30 minutes. Although the overall analysis time is dominated by file I/O, the new approach is orders of magnitude faster than the corresponding sequential analysis carried out by the EXPERT tool, thereby enabling analyses at scales that have been previously inaccessible. A screenshot with analysis results for 16,384 processes is shown in Figure 3.

References

[1] **Wolf, F., Mohr, B.**
Automatic performance analysis of hybrid MPI/OpenMP applications,
Journal of Systems Architecture 49(10-11), pp. 421-439, 2003

[2] **http://www.scalasca.org**

[3] **Geimer, M., Wolf, F., Wylie, B.J.N., Mohr, B.**
Scalable Parallel Trace-Based Performance Analysis, Proceedings EuroPVM/MPI 2006, Springer LNCS 4192, pp. 303-312, 2006

• Markus Geimer[1]
• Felix Wolf[1,2]
• Brian J.N. Wylie[1]
• Bernd Mohr[1]

[1] John von Neumann Institut für Computing (NIC) Forschungszentrum Jülich

[2] Fachgruppe Informatik RWTH Aachen